程序设计基础大作业-飞行棋

文档修改说明

本文档目前经过两次修改，第一版修改内容**都用红色字体标注**，第二版修改内容**都用蓝色字体标注**，请注意查看文档中这部分的内容，如果可以的话请相互提醒。

学术诚信

所有课程作业都需要**独立完成。**

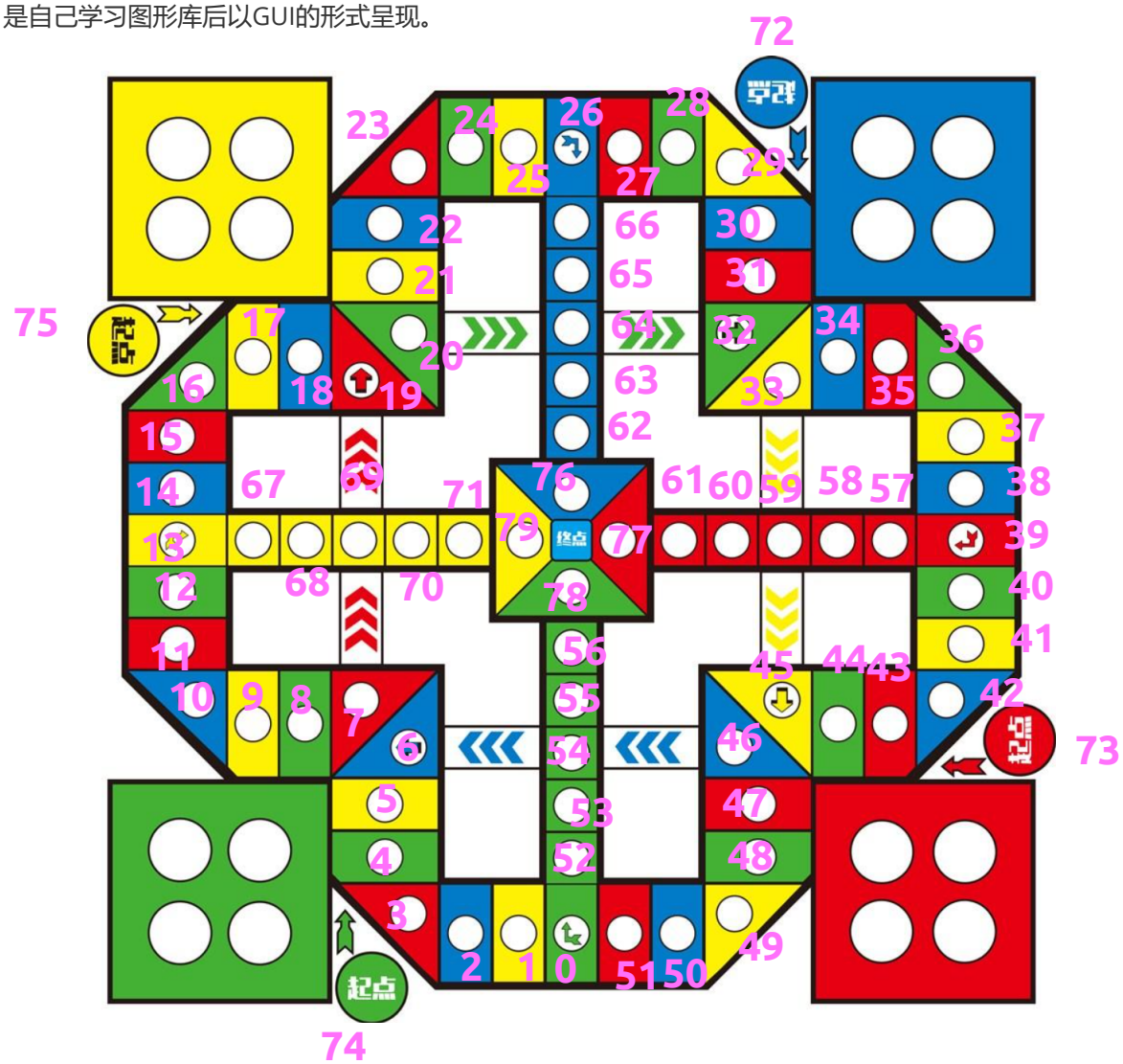
详情请参考 MIT 对学术诚信的解释。请不要将你的代码以任何形式公开发布或给他人传阅，这么做可能导致代码相似被判定为抄袭。

如果对于本项目有任何疑问，可以与你课程的助教或发邮件与191250114@smail.nju.edu.cn进行沟通。

项目目标

飞行棋是十字戏类游戏，以模拟飞机飞航为主题，游戏以飞机由机场起飞至目的地，所以称为飞行棋。飞行棋是中国参考英国十字戏发展出来的，而英国十字戏是从印度十字戏演变出来的。

在这个项目里，需要大家完成一个简易版本的飞行棋游戏，最终呈现的效果可以是命令行呈现，也可以是自己学习图形库后以GUI的形式呈现。



项目规则

实现方式

限定语言为c/c++，对于第三方图形库不作限制。最终实现的效果既可以是通过在终端里打印飞行棋的棋盘来表现，也可以是完整的图形界面。但要求一定要对棋盘上的情况有所呈现。

简化版飞行棋规则(游戏基本逻辑)

> **有多个但不少于2个“玩家”**

每个“玩家”操控一个颜色对应的棋子，“玩家”可以是真人控制也可以是电脑控制，但要求每局游戏至少要有一个真人玩家和一个电脑玩家。真人通过与实现系统交互来进行游戏，电脑玩家自动进行游戏。

开场的时候通过输入数字自定义玩家的数目，同时输入对应选项确定每个玩家类型，但是至少要一个电脑和一个真人，不推荐玩家的数量超过4个

> **每个玩家四颗棋子，初始都在机场不能出门，按顺序投骰子（1-6点的骰子）。如果玩家骰子抛出6，有以下几个情况：**
>
> 1. **（如果有棋子仍在机场）让自己任意一个在机场的棋子在起点准备出发，并且再抛一次骰子决定可以出门走几格（该次抛骰子无论抛出几都只能作为刚刚准备出发的棋子前进的步数）。**
> 2. **(所有棋子都已经出发不在机场)可以选择任意一个没到达终点的棋子前进6步**
> 3. **(如果有棋子在机场有棋子已经出发)依然只能使用情况1的方案选择一个棋子出发，而不能让已经出发的棋子移动**

如果玩家的每个棋子都在机场或者终点处，且没有抛到6，则该玩家这个回合结束。

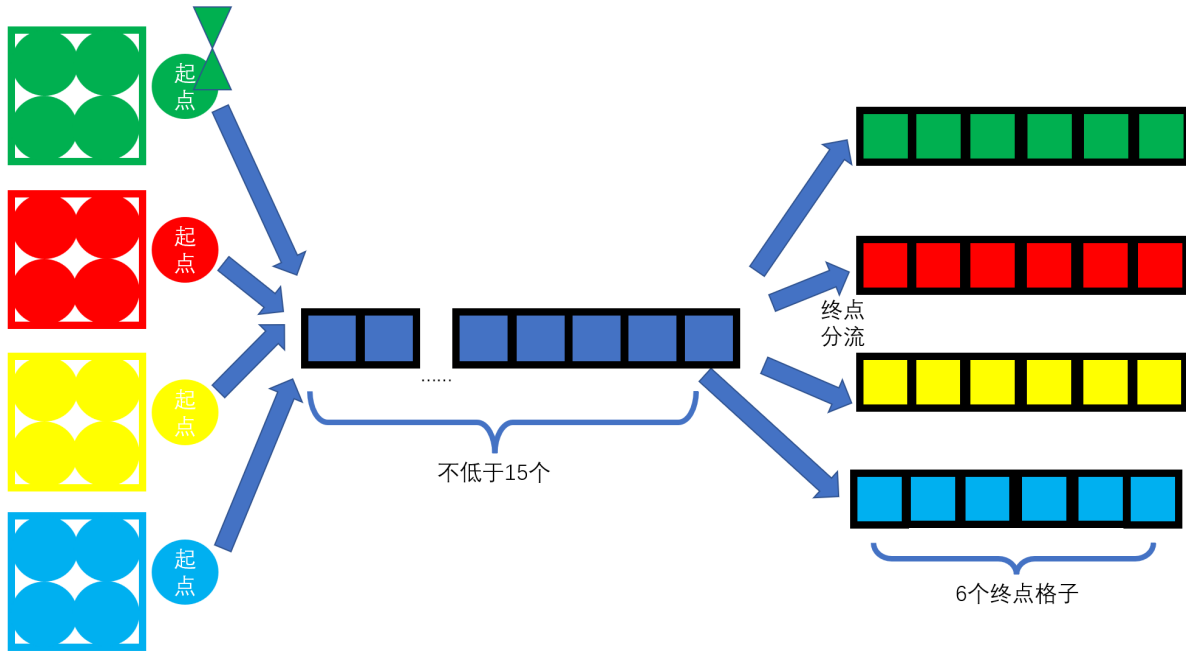
> **如果玩家抛出的点数是1-5，且有至少一个既不在机场也不在终点的棋子，则可以选择一个棋子根据点数往前移动一定步数。在简化实现中，所有的棋子都共享一条直线型的跑道！**

为了方便大家的实现，在这里做了简化处理。如果有同学想实现原版的效果，可以作为扩展功能实现。

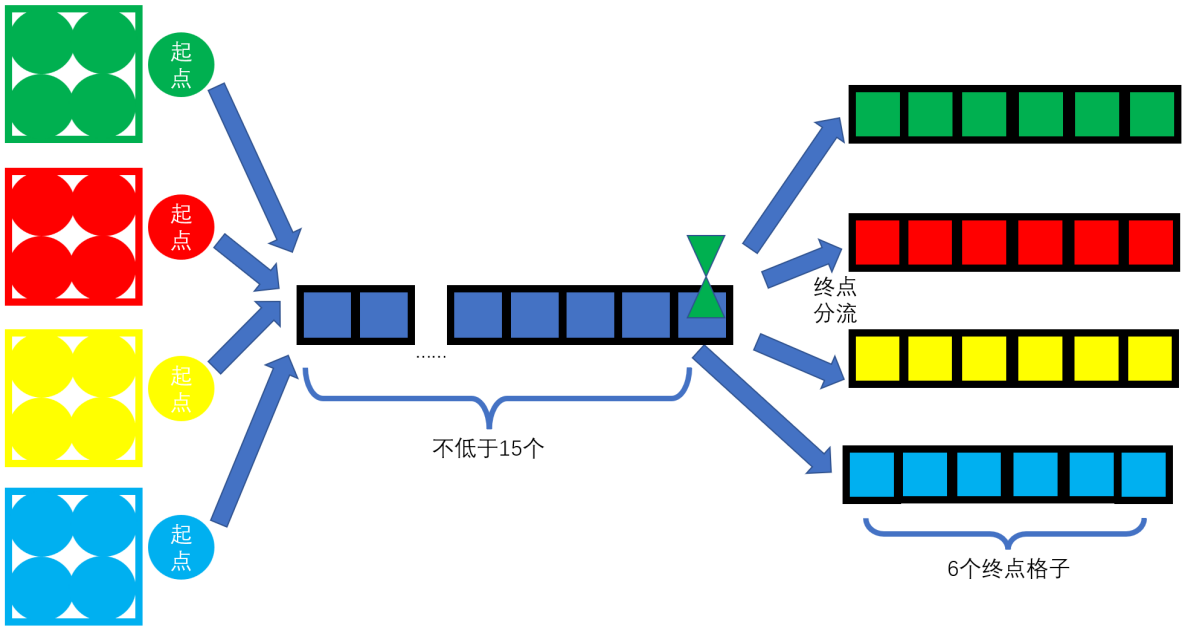
同时要求 **直线型的跑道格子数量不低于15个**

也就是下图中中间的深蓝色格子不少于15个。

此外，终点格子的数量应当为6个

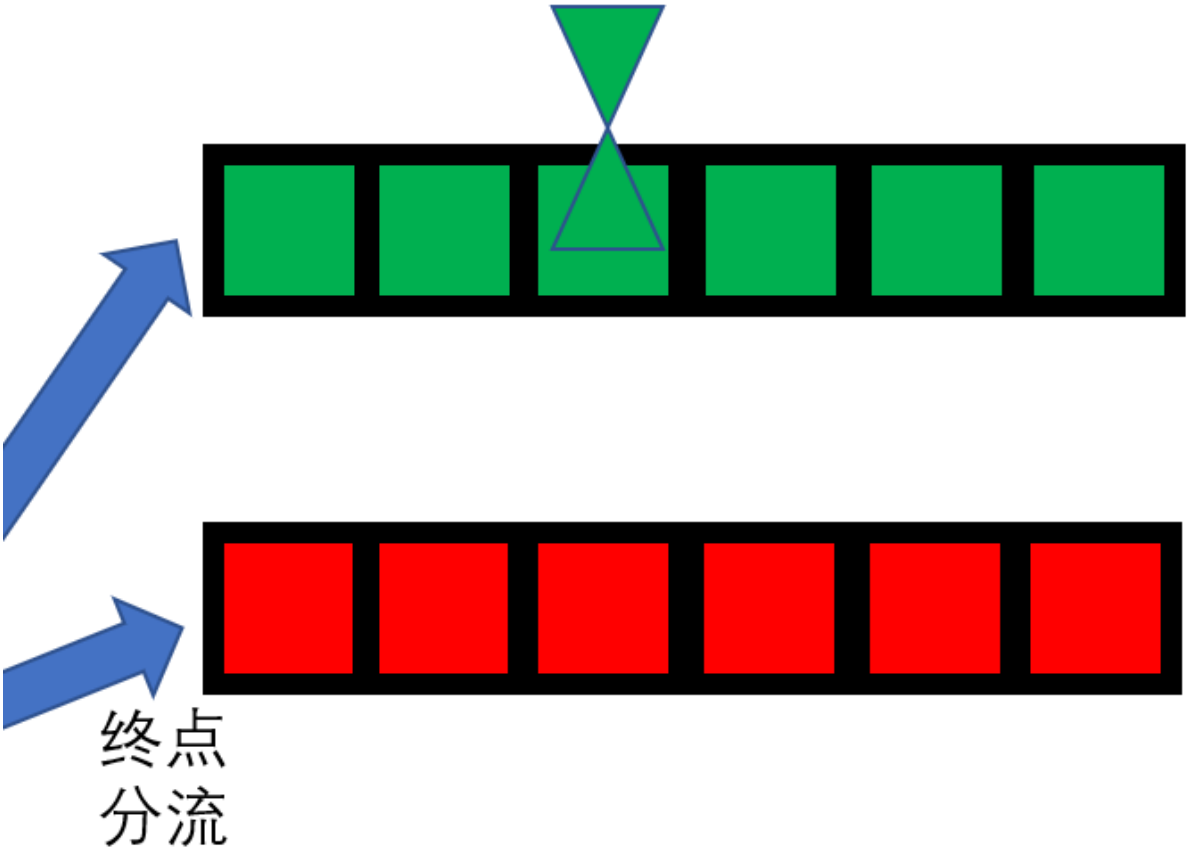


> **在终点前要进行分流，各自进入对应颜色的分流终点。然后进入终点分流阶段。**

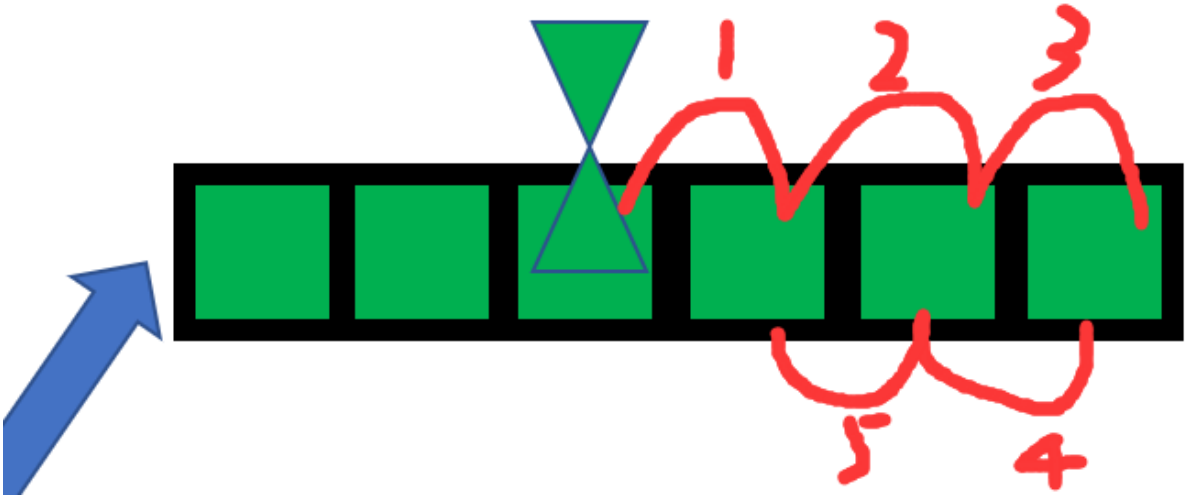


如图，绿色的棋子如果此时是抛出了某些点数，或者是因为前面抛出了点数走到了这个位置还有几步没有走完，接下来都应该进入绿色对应的终点区域

> **在终点区域，到达终点的判定是正好走到最后一格。如果没有走到则等待下一轮抛投，如果走到最后一格还有剩余步数，则需要反弹剩余步数。**



如上图，在当前位置时，如果抛出的点数是3，则正好到达终点。如果抛出的点数是5，则反弹效果如图所示。



> **一颗棋子达到终点的时候，就将这个棋子移出整个棋盘，记为成功到达。玩家胜利的条件是让全部的四颗棋子都到达终点。**
>
> **只要有一个玩家的四颗棋子到达终点，游戏就立刻结束**
>
> **对于同格子情况的处理：有不同的规则版本，在此处统一。**
>
> - 对于途径的其他棋子无视，不论那个途径的格子里有几个棋子。
> - 如果本次移动最终落点处是己方棋子，则这两个棋子可以同时存在这个格子里（但是不采取某些可以叠子的规则，后续再进行移动的时候同一格的棋子仍然视为多个分开的个体前进）
> - 如果本次移动最终落点处有敌方棋子，则己方棋子占据这个格子，所有在这个格子的敌方棋子不论数量都返回机场（也就是需要抛到6才能再出发的状态）

电脑操作

对于上述提到的电脑玩家操作，应当保证电脑的表现正常，即不存在非法操作（比如没有投到6但是让一个棋子出发），而且所有操作都是正常操作（比如不会在投了骰子且有棋子可以移动的情况下什么都不做）。

电脑玩家的操作也应当以某种方式（输出日志或其他）展现出来。

扩展功能

上述描述的实现标准是一个简化改进版本的要求，同学们可以在现有的基础上进行扩展，包括但不限于：

- 一个实现的非常好的GUI或者交互系统
- 体现简化版本中没有提到的“飞”，即在某个满足特定要求的特定位置时可以飞到另外一格
- 体现简化版本中没有提到的“跳”，即给中间的深蓝色格子进行染色，当棋子移动的最终落点在同色的格子上时可以往前跳到下一个同色格子上。
- 把跑道改为原版中的环形跑道的设计
- 文档里各处提到可以作为扩展功能的实现
- ……**不在要求之内，但是合理的，都当作扩展功能**

可以在各处进行拓展，**但是理论上不能破坏现在的基本规则或者是使得规则变得更简单改动**（例如删除终点的反弹但是又不设计一个更复杂的规则代替都不被认为是拓展）

DEBUG模式

为了方便助教测试和自己debug，推荐自己在程序里加入一个“外挂”，比如在输入某些指令后，或者全局定义某些变量值为1或自定义宏(例如#define DEBUG)开启自己定义的debug模式，你可以编写操控骰子投出的点数的代码，从而可以控制棋子的走动。

示例代码：

```
#define DEBUG
//假设这是一个用于得到投骰子结果点数的函数
int getDieRes(){
    #ifdef DEBUG
    //TODO
    #else
    //TODO
    #endif
}
```

该部分不计入分数，但是强烈推荐做一下。

实现步骤提示

声明

该部分仅作为参考的建议和提示，如果你有更好的设计思路和实现，完全可以自己采用，并可以在项目报告中进行展现。

状态

对于棋子和棋盘格状态的存储，用数组和变量的方式就可以实现。比如用数组来记录当前棋子的位置和当前的状态，再用一些数组来存储棋盘格子内的状态。

可扩展的方向：

利用结构体+指针的方式来实现，这样的设计可以被认定为扩展功能一部分的分数。

```
typedef struct square {
    int type;//标注这个格子是出生的格子还是中间的格子还是终点阶段的格子等
    struct square* next;//下一个格子
    int cur_plane_type;//现在站在这个格子上的棋子的颜色
    int cur_plane_num;//现在站在这个格子上的棋子的数量(因为一个格子上不会有两种阵营的棋子)
    //……可以添加更多的属性
}square;

typedef struct plane {
    int type;//棋子所属阵营
    square* cur_pos;//现在在哪个格子上
    //……可以添加更多的属性
}plane;
```

游戏控制

对于游戏整体逻辑控制 可以参考如下的设计

```
while(!game_ends){
    player=(player+1)%player_num;//切换玩家
    die_point=getDieRes();//投骰子
    move(player,die_point);//移动部分的判断
    game_ends=judge();//判断游戏是否结束
}
```

输出

如果选择CLI：

只是简单的将模拟的棋盘用一些代表特殊意义的字符不断的输出在屏幕上是可以的。例如像下面的模拟输出：

注：这只是一个可以参考的字符界面输出方式，对于叠子的情况也只是参考

```
--------------------------------------------------
GG
GG                          DDDDD(green)

  □□□□□□□□□□□□□□□□□□□□□□□□□□

YY                          DDDDD(yellow)
YY
Green team throw the die and get 2 point!
Green team ubable to move!
--------------------------------------------------
GG
GG                          DDDDD(green)

  □Y□□□□□□□□□□□□□□□□□□□□□□□□

Y□                          DDDDD(yellow)
YY
Yellow team throw the die and get 6 point!
Yellow team throw the die again and get 2 point!
Yellow team moves.
--------------------------------------------------
/*
对于叠子的情况，可以试试在方块下面输出一个数字来表索这个格子上叠了几个子。
而且由于规则设计，一个格子里是不会同时出现两个不同阵营的棋子的。
*/
GG
GG                          DDDDD(green)

  □Y□□□□□□□□□□□□□□□□□□□□□□□□
   2
□□                          DDDDD(yellow)
YY
```

当然，如果你对字符进行了染色并且实现了刷屏的效果(屏幕上不是通过类似上述分隔符分隔的多次输出来展示，而是通过清屏再输出或者修改屏幕上显示的某些字符来实现)，这个设计可以被认为是一个扩展功能，根据效果可以作为评分标准里扩展功能部分的分数。

对于没有接触过CLI的同学，下面会给出一个命令行交互的简要介绍。

命令行交互简要介绍

如果选择GUI

对于会使用GUI的同学，在设计上应该不需要我们给出什么建议。对于没有接触过但想尝试的同学，可以试着了解一下SDL2，Qt，Easyx等图形库，但这些库的上手程度相对会较高，我们不鼓励同学内卷，故在此处不多做介绍，有兴趣的同学自行了解。

由于使用图形库会使得代码量显著变大，故采用GUI的设计可以视为很好的实现了拓展功能的部分。

先动脑 再动手!

由于在很多地方都存在共通之处，同学们在编写代码的时候可以先思考一下有些地方的代码是不是可以抽象出来作为一个函数在多个地方复用，而通过设置一些变量来进行区分，这样可以很好的实现代码的压缩，防止出现屎山，也会减少在复制的时候有些地方没改完全导致出错!

对于每个玩家阵营，很多流程是一样的，是否可以把里面的一些操作抽象出来，而不是在每回合的循环里复制四遍代码?

比如对于每个棋子（飞机）可以增加一个标签来标记他是什么颜色的棋子，这样对于棋子的一些操作就可以通用了。

比如对于真人和电脑玩家的区分，本质区别其实只有在移动的时候有区别，而且移动都要检查是否合法，那么可以考虑一个设计是：对于玩家的每个棋子，判断投出骰子对应点数的移动是否合法，如果合法，那么就返回一个所有可以选择移动的棋子列表，电脑是从中随机选一个移动，而真人是自己选择。这样真人和电脑的代码差异就变得很小了。

评分标准

测试(90%)

- 每个人五分钟的时间，展示实现的功能。
- 分数分布：

1. 展示游戏的基本逻辑实现----------55%
2. 展示电脑玩家的自动操作----------10%
3. 展示扩展功能----------10%（10%为上限，根据拓展效果给分）
4. 良好的代码框架设计设计----------10%（体现在模块化程序设计、功能分解和复合、过程抽象等等基本思想的使用）
5. 代码风格----------5%(函数和变量的命名，格式缩进等)
6. 展示过程中如果出现bug，酌情扣分

报告(10%)

在报告中可以描述你的实现，你认为设计的比较好的地方等等，但不要大段的粘贴你项目里的代码实现。我们不鼓励内卷，所以推荐你的报告不要超过两页。

验收流程

助教根据学生提交的源码和可执行文件尝试编译运行，如果无法运行，会联系同学线上或线下验收

- **因此，推荐使用例如ncurses或qt等通用库**
- **qt支持跨平台，但是需要自学makefile**
- **如果使用了第三方库，推荐自行针对自己选择的库查阅如何生成release版本的可执行的文件（也就是打包发布的方法，而不是只是单纯在提交的压缩包里复制一个编译生成的exe，这样的文件多半是无法在另外一台电脑上运行的）**

提交方式

将源码的压缩包提交到课程系统中，保持原始的目录结构，并在文档中说明如何你的源码如何编译生成对应的可执行文件和基本操作的方式

可以参考如下的结构

```
|--学号_姓名.zip
|   |--src //(源码文件夹)
|   |   |--a.cpp
|   |   |--b.cpp
|   |   |--Makefile
|   |   |--……
|   |--bin //(可执行文件文件夹，如果有)
|   |   |--……
|   |--实验报告_学号_姓名.pdf
```